

	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2
--	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---

5
10
15
20
25
30
35
40
45
50
55

- a. + - * / % +(unary) -(unary)
- b. += -= *= /= %= ++ --
- c. & | ^ << >> ~
- d. &= |= ^= <<= >>=
- e. < > <= >= == != !

5

5. No functions have the virtual label in order to remove the extra level of indirection in the V-Table. Only add them if I decide to inherit more types.

10

6. Class behavior is undefined if there are more than $2^{(\text{sizeof(int)} * \text{BITS_PER_BYTE} - 1) - 4}$ bits in the number. Ways around even this limit (though you'll run out of RAM before reaching it) involve using the STL array class instead of IntNode lists, and using LargeInt numbers internally. Both of which reduce the performance and speed of the datatype enough that it's not worth it.

15

7. The class functions are all made such that no datatype size information is hardcoded. Everything is determined at runtime/compiletime to allow the program to run on computers where int is not 32-bits (ie, 36 or 64-bit computers). The only thing that is hardcoded is BITS_PER_BYTE. This can be changed and the rest of the functions should still behave correctly (assuming BITS_PER_BYTE remains accurate). One restriction is that built-in datatypes (int, short, etc) much be a multiple of bytes (ie $\text{BITS_PER_DATATYPE} \% \text{BITS_PER_BYTE} = 0$).

20

8. The library is OS independent, and adheres to the Standard C++ Library. Any half-decent compiler should be able to compile it. GCC < 3.0 is not a half-decent compiler, LargeInt will only compile on GCC >= 3.0.

25

9. Computations are performed using int type, while the values are stored in INTNODE_TYPE. To change the storage type, simply redefine INTNODE_TYPE. Intrinsic type int must be a multiple of INTNODE_TYPE and at least twice the size of INTNODE_TYPE. Also, INTNODE_TYPE must be an unsigned type.

30

10. The internal data structure is a bi-directional list of IntNodes. A Most Significant Bit and Least Significant Bit pointer are used to access it. These will be null if the number is 0. In order to greatly speed up the creation of these lists (especially with the heavy reliance on temporary instances), IntNode overloads the new and delete operators and stores a pool of previously used nodes. So new and delete calls will only take half a dozen operations, instead of having to go through malloc and free. This gives an experimental speedup of 5.3x.

35

11. In order for bitwise operations to make sense, I had to add in preliminary fixed bit support. If fixed bit is off, ~, Twos, and ^ will do nothing. & and | will only work if all operands are positive, otherwise they will do nothing. If FixedBit is On, The ~, Twos, and ^ operators will truncate/expand the inputs to the FixedBitLength, convert to unsignedness, perform the operation on that specified bit length, and then return the usual result. & and | will do that only if either operand is negative. On converting to (const char *), the result will be truncated/expanded to the desired bit length and converted to unsignedness.

40

12. In order to reduce the overhead of temporaries, I implemented "copy by stealing". With this, the IntNode list (the number) from a temporary is stolen $O(1)$ instead of copied $O(n)$.

45

End User Notes

50

1. LargeInt is a datatype. This means you use it just like you would any other datatype. It has a superset of "int" functionality. Anything you do with an int variable, you can do with a LargeInt variable.

2. Additions to "int" functionality are Sign(), Twos(), and Divide() functions

3. Efficiency notes:

55

- a. Unary operations are optimized for efficiency and speed. ++X is faster than X++, both are faster than X += 1. Use ++X/--X whenever possible.
- b. if(X) and if(!X) are faster than comparing to zero.
- c. Construction is faster than assignment (it involves one less temporary copy).

- d. Constructing from a decimal number (const char *) is slow. Use Hex, Oct, or Bin.
- e. Outputting in decimal (const char *) is slow. Use Hex, Oct, or Bin. Specify ios::hex to the ostream class, or set the LargeIntBase flag and call (const char *) before ostream<<.
- f. Division and multiplication are the slowest operations. But that's nothing new. Remember that division and modulo can be performed at the same time with Divide().
- g. If a constant is used more than once, construct a constant LargeInt to hold it. Otherwise a LargeInt will be constructed from it anyway everytime the constant is used with a LargeInt expression.
- h. Use (-) instead of Twos() for regular two's compliment, unless you desire an unsigned result. (-) is faster.
4. The only storage that maintains full precision is the character array. All other conversions are potentially lossy. The LargeInt can be serialized using ofstream << and initialized using ifstream >>, as well as from user input (o/istream, o/istrstream).
- a. Note that using routines from math.h will cause potentially lossy conversions to take place. If you have to, convert to double, as this will retain the 53 most significant bits of a number 1023 bits long.
5. The library is not thread aware. It is up to the user to make sure a LargeInt is only accessed once at a time, the same as any other variable.
6. Changing the flags will affect all LargeInt variables.
- a. Set/GetFlags() See documentation below.
- b. Set/GetPrePostFix() See documentation below
- c. You should push/pop any flags you change in a function to avoid unexpected side-effects in the calling function.
- i. Example code:
- ```
foo()
{
 //Store old state info and set new info
 unsigned int OldFixedBit =
 GetFlag(LargeInt::FixedBit);
 SetFlag(LargeInt::FixedBit, 36);
 LargeInt::BaseEnum OldBase =
 GetFlag(LargeInt::PrintBase);
 SetFlag(LargeInt::PrintBase,
 LargeInt::Bin);
 //Store old pre/postfix for binary output //and set new
 string OldPrefix, OldPostfix;
 GetPrePostFix(OldPrefix, OldPostfix);
 SetPrePostFix("1\\'", "\\");

 //Restore old state
 SetPrePostFix(OldPrefix, OldPostfix);
 SetFlag(LargeInt::PrintBase, OldBase);
 SetFlag(LargeInt::FixedBit, OldFixedBit);
}
```
- ii. Note that PrintBase and Pre/Postfix only apply to (const char \*) conversion, so you usually won't need to care about those.
7. Make sure LargeInt::FixedBit is set appropriately before performing operations on LargeInt variables.
- a. If Fixed Bit is turned off
- i. ~, ^, and Twos() will return 0 if FixedBit is turned off.
- ii. & and | will return 0 if FixedBit is turned off and either operand is negative. If both operands are positive they will perform infinite precision & and |.
- iii. LargeIntUnsigned will be turned off.

b. If Fixed Bit is turned on

i. ~, Twos(), ^, and (const char \*) conversion will unsign/ truncate the result to the fixed bit length. & and | will do that if either operand is negative.

ii. Be aware of this if using | for quick addition. | is not significantly faster than + on LargeInts.

c.

8. Do not store the return value from (const char \*). The return value will become an invalid pointer the next time (const char \*) is called on any LargeInt. Use strcpy to save the result, or construct a C++ string or iostream off of it.

```

/*-----+++++,,,,,,,,,,,,,,+++++-----*/
class LargeInt
{
15 protected:

 /** Internal Data Structures */

 //List node structure (bi-directional)
20 template<class T>
 struct _TIntNode
 {
 //Constructors
 _TIntNode(); //m_Value = 0;
25 _TIntNode(const T &); //m_Value = uint
 _TIntNode(const _TIntNode &);
 //The operator= only copies the value.
 //The next/prev pointers remain unchanged
 _TIntNode &operator=(const _TIntNode &);

30 //Member Data.
 T m_Value;
 _TIntNode *m_pPrev;
 _TIntNode *m_pNext;
35 typedef T Type;

 //Decrease the allocation/deallocation overhead of the
 //dynamic node list by providing a node pool to enable
 //reuse of node memory.

40 //Points to the head of the pool list (singly-linked)
 static _TIntNode *m_pNodePool;
 //The number of currently existing owners. An owner is any
 //instance that could potentially use an IntNode. If no //more owners exist, we
45 can free up all the memory in the //node pool.
 static unsigned int m_OwnerCount;
 //The number of nodes in the pool.
 static unsigned int m_NodeCount;
 //The number of nodes allowed in the pool per owner.
50 //Increase the value to have fewer //allocations/deallocations. Decrease the
 value to conserve //more run-time memory.
 #define NODEPOOL_RATIO 4
 //Controll alloc/dealloc
 static void *operator new(unsigned int);
55 static void operator delete(void *);
 };

```

public:

```
//Flag Enumerations
enum BaseEnum {Hex = 16, Dec = 10, Oct = 8, Bin = 2};
enum OnOffEnum {On = 1, Off = 0};
enum FlagEnum {PrintBase, FixedBit};
```

protected:

```
//Local defines
typedef _TIntNode<INTNODE_TYPE> IntNode;
#define BITS_PER_INT (sizeof(unsigned int) * BITS_PER_BYTE)
#define BITS_PER_NODE (sizeof(INTNODE_TYPE) * BITS_PER_BYTE)
#define INT_MSB_MASK (1 << (BITS_PER_INT - 1))
#define INTNODE_MSB_MASK (1 << (sizeof(INTNODE_TYPE) * BITS_PER_BYTE -
```

```
1))
//This is simply the point at which I can no longer count the
//number of bits a LargeInt is holding using an integer. Knowing
//the exact number of bits a LargeInt holds is necessary for //several functions. Also,
indexing is required for const char * //conversion.
#define MAX_NODES (((unsigned int)(-1) >> 1) - 4) / BITS_PER_NODE)
```

```
/** Internal List Methods **/
```

```
//The list will always be compacted. Which means any funtions
//that might alter the list will call Compact before returning. //A LargeInt who's value is
0 will have an empty list,
//and will be positive.
```

```
//The head (LSB) and tail (MSB) of the list.
```

```
IntNode *m_pLSB, *m_pMSB;
```

```
//Number of nodes.
```

```
unsigned int m_NodeCount;
```

```
//Insert a node (nodes) at the end with value = Value
```

```
void Expand(unsigned int);
```

```
void Expand(IntNode::Type = 0);
```

```
//Remove nodes from the end that are no longer needed
//Since compact must always be called anytime the list changes,
//compact will take care of assuring there is never a negative
//zero.
```

```
void Compact();
```

```
//Truncate to FixedBit # bits. Do nothing if FixedBit isn't
```

```
//turned on.
```

```
void Truncate();
```

```
/** Internal Helper Functions **/
```

```
//Helpers for parsing the char*. The first "unsigned int" is the
```

```
//string length
```

```
//The second "int" is the index into char *
```

```
//Looks for + or -
```

```
//Returns false if no more characters in the string.
```

```
//Returns the index to the character after the sign in "int &"
```

```
bool ParseSign(const char *, int, int &);
```

```
//Looks for 0x, 0, h, x, o, or b (upper or lower case)
```

```
//Sets BaseEnum to be Hex, Oct, Dec, or Bin based on the prefix
```

```

//Returns false if no more characters in the string.
bool ParseBase(const char *, int, int &, BaseEnum &);
//Parses a hex number starting at index "int"
//Returns the index to the first character not parsed.
5 int ParseHex(const char *, int, int);
 int ParseDec(const char *, int, int);
 int ParseOct(const char *, int, int);
 int ParseBin(const char *, int, int);

10 //Common code for constructing from integer
 //bool=true means signed, false means unsigned
 void FromInt(bool, unsigned int);
 //Common code for converting to integer
 unsigned int ToInt() const;
15 //Common code for constructing from floating point
 void FromFloat(long double);
 //Common code for converting to floating point
 enum FloatEnum {Float, Double, LongDouble};
 long double ToFloat(FloatEnum) const;

20 //Converts a signed LargeInt to a fixed bit unsigned number. This
 //result will only be used for printing purposes or as a
 //temporary in binary operations. It is no longer valid if //fixedbit changes, and cannot
 //be returned to signed format.
25 friend LargeInt UnSign(const LargeInt &);

 /*** Flag Variables ***/

 //false = positive, true = negative
30 bool m_Sign;
 //false = regular signed int. true = this is the output of
 //UnSign(). It is only inherited if m_CopyByStealing is enabled.
 bool m_UnSigned;

35 static unsigned int m_FixedBit;
 static BaseEnum m_Base;

 static string m_HexPrefix;
 static string m_HexPostfix;
40 static string m_DecPrefix;
 static string m_DecPostfix;
 static string m_OctPrefix;
 static string m_OctPostfix;
 static string m_BinPrefix;
45 static string m_BinPostfix;

 //Enabling this can vastly speed up copy construction, but should
 //only be used when you know you're copying a temporary which
 //will be destroyed shortly anyway. Otherwise you'll lose data.
50 //false = perform full copy on each node
 //true = steal the nodes because it's just a temporary
 //This only affects the next call to operator=(const LargeInt &)
 //or LargeInt(const LargeInt&). They reset this flag to false.
 static bool m_CopyByStealing;

55 public:

```

```

//*** Constructors ***

//Default, initialized pointers to NULL
LargeInt();
5 //Constructs large int from bool, signed/unsigned char,
//short, int, and long
LargeInt(signed int);
LargeInt(unsigned int);
10 LargeInt(signed long);
LargeInt(unsigned long);
//Constructs large int from float and double
LargeInt(double);
LargeInt(long double);
//Constructs large int from constant (char array). The constant
15 //will be assumed to be signed. The number will be extracted from //the MSB until the
first formatting error or EOS.
//Prefixes:
// The standard C prefixes ox(Hex) and o(Oct) will be
//accepted, as well as h(hex), x(hex), o(oct), and b(bin) (upper
20 //or lower case). The sign (-/+ /nothing) must come before the //prefix. Converting from
decimal is slow. Make your constants in //hex, oct, or bin format when possible.
LargeInt(const char *);
//Copy constructor
LargeInt(const LargeInt &);
25

//Destructor.
~LargeInt();

//Assignment Operator. Everything is converted to LargeInt before
30 //assignment is called.
LargeInt &operator=(const LargeInt &);

//*** Flag Functions ***

35 //Set/get the internal flags to value uint.
//Supported flags are:
// LargeIntBase: Hex, Dec, Oct, Bin - for (const
//char *) conversion
// LargeIntFixedBit: Off, #- If Off, turns off FixedBit
40 //operations. - If #, turns on
//
//FixedBit operations. Specifies # bits used for bitwise
//operations (~, &, |, ^) and for conversion (to int, double, and
//const char *).
45 static void SetFlag(FlagEnum, unsigned int);
static unsigned int GetFlag(FlagEnum);

//Set/Get the prefix and postfix used for printing the LargeInt
//in the specified base.
50 //BaseEnum = Hex, Dec, Oct, Bin. Strings can be anything. const
//char * will print (+/-)Prefix####Postfix. Default Prefixes and
//postfixes are already defined.
static void SetPrePostFix(const string &, const string &,
BaseEnum = m_Base);
55 static void GetPrePostFix(string &, string &, BaseEnum = m_Base);

//Returns true if this is negative, else false;

```

```

 bool Sign();

 /*** Conversions ***

5 //Convert a LargeInt to bool, signed/unsigned char,
 //short, int, and long
 operator bool() const;
 operator char() const;
 operator signed char() const;
10 operator unsigned char() const;
 operator signed short() const;
 operator unsigned short() const;
 operator signed int() const;
 operator unsigned int() const;
15 operator signed long() const;
 operator unsigned long() const;
 //Convert a large int to float or double.
 operator float() const;
 operator double() const;
20 operator long double() const;
 //Print the number in character format. Printing in decimal is
 //slow. Print in hex, oct, or bin if possible.
 //Do not store the return value. The return value will become an
 //invalid pointer the next time (const char *) is called on any
25 //LargeInt. Use strcpy to save the result, or construct a C++
 //string or iostream off of it.
 operator const char *() const;

 /*** Operator Macros ***
30 #define OPDEFN(Op, LType, RType) friend LargeInt operator Op(LType,
 RType);
 #define L const LargeInt &

 /*** Arithmetic ***
35
 OPDEFN(+, L, L) OPDEFN(+, L, bool)
 OPDEFN(+, L, char) OPDEFN(+, L, const char *)
 OPDEFN(+, L, signed char) OPDEFN(+, L, unsigned char) OPDEFN(+, L,
short) OPDEFN(+, L, unsigned short)
40 OPDEFN(+, L, signed int) OPDEFN(+, L, unsigned int) OPDEFN(+, L,
long) OPDEFN(+, L, unsigned long)
 OPDEFN(+, L, float) OPDEFN(+, L, double) OPDEFN(+, L,
long double) OPDEFN(+, bool,L) OPDEFN(+, char,L)
 OPDEFN(+, const char *, L)
45 OPDEFN(+, signed char,L) OPDEFN(+, unsigned, char,L) OPDEFN(+, short,
L) OPDEFN(+, unsigned short,L)
 OPDEFN(+, signed int,L) OPDEFN(+, unsigned int, L)
 OPDEFN(+, long,L) OPDEFN(+, unsigned long,L)
 OPDEFN(+, float, L) OPDEFN(+, double, L)
50 OPDEFN(+, long double, L) OPDEFN(-, L, L)
 OPDEFN(-, L, bool) OPDEFN(-, L, char) OPDEFN(-, L, const
char *) OPDEFN(-, L, signed char) OPDEFN(-, L, unsigned char) OPDEFN(-, L, short)
 OPDEFN(-, L, unsigned short) OPDEFN(-, L, signed int) OPDEFN(-, L,
unsigned int) OPDEFN(-, L, long) OPDEFN(-, L, unsigned long)
55 OPDEFN(-, L, float) OPDEFN(-, L, double) OPDEFN(-, L,
long double)
 OPDEFN(-, bool,L) OPDEFN(-, char,L)

```



|    |                                          |                              |                            |
|----|------------------------------------------|------------------------------|----------------------------|
|    | OPDEFN(-, const char *,                  | L)                           |                            |
|    | OPDEFN(-, signed char,L)                 | OPDEFN(-, unsigned char,     | L) OPDEFN(-,               |
|    | short,L)                                 | OPDEFN(-, unsigned short,    | L)                         |
| 5  | OPDEFN(-, signed int,L)                  | OPDEFN(-, unsigned int,      | L)                         |
|    | OPDEFN(-, long,L)                        | OPDEFN(-, unsigned long,     | L)                         |
|    | OPDEFN(-, float,L)                       | OPDEFN(-, double,            | L)                         |
|    | OPDEFN(-, long double,L)                 | OPDEFN(*, L, L)              |                            |
|    | OPDEFN(*, L, bool)                       | OPDEFN(*, L, char)           | OPDEFN(*, L,               |
|    | const char *)                            |                              |                            |
| 10 | OPDEFN(*, L, signed char)                | OPDEFN(*, L, unsigned char)  | OPDEFN(*, L,               |
|    | short) OPDEFN(*, L, unsigned short)      |                              |                            |
|    | OPDEFN(*, L, signed int)                 | OPDEFN(*, L, unsigned int)   | OPDEFN(*, L,               |
|    | long) OPDEFN(*, L, unsigned long)        |                              |                            |
|    | OPDEFN(*, L, float)                      | OPDEFN(*, L, double)         | OPDEFN(*, L,               |
| 15 | long double)                             |                              |                            |
|    | OPDEFN(*, bool,L)                        | OPDEFN(*, char,L)            |                            |
|    | OPDEFN(*, const char *,                  | L)                           |                            |
|    | OPDEFN(*, signed char,                   | L)                           | OPDEFN(*, unsigned char,L) |
| 20 | short, L) OPDEFN(*, unsigned short,L)    |                              |                            |
|    | OPDEFN(*, signed int,L)                  | OPDEFN(*, unsigned int,      | L)                         |
|    | OPDEFN(*, long,L)                        | OPDEFN(*, unsigned long,L)   |                            |
|    | OPDEFN(*, float,L)                       | OPDEFN(*, double,            | L)                         |
|    | OPDEFN(*, long double,L)                 | OPDEFN(/, L, L)              |                            |
|    | OPDEFN(/, L, bool)                       | OPDEFN(/, L, char)           | OPDEFN(/, L,               |
| 25 | const char *)                            |                              |                            |
|    | OPDEFN(/, L, signed char)                | OPDEFN(/, L, unsigned char)  | OPDEFN(/, L,               |
|    | short) OPDEFN(/, L, unsigned short)      |                              |                            |
|    | OPDEFN(/, L, signed int)                 | OPDEFN(/, L, unsigned int)   | OPDEFN(/, L,               |
|    | long) OPDEFN(/, L, unsigned long)        |                              |                            |
| 30 | OPDEFN(/, L, float)                      | OPDEFN(/, L, double)         | OPDEFN(/, L,               |
|    | long double) OPDEFN(/, bool,L)           |                              | OPDEFN(/, char,L)          |
|    | OPDEFN(/, const char *,L)                |                              |                            |
|    | OPDEFN(/, signed char,L)                 | OPDEFN(/, unsigned char,L)   | OPDEFN(/, short,L)         |
|    | OPDEFN(/, unsigned short,L)              |                              |                            |
| 35 | OPDEFN(/, signed int,L)                  | OPDEFN(/, unsigned int,      | L)                         |
|    | OPDEFN(/, long,L)                        | OPDEFN(/, unsigned long,L)   |                            |
|    | OPDEFN(/, float,L)                       | OPDEFN(/, double,            | L)                         |
|    | OPDEFN(/, long double,L)                 | OPDEFN(% , L, L)             |                            |
|    | OPDEFN(% , L, bool)                      | OPDEFN(% , L, char)          | OPDEFN(% , L, const        |
| 40 | char *) OPDEFN(% , L, signed char)       | OPDEFN(% , L, unsigned char) | OPDEFN(% , L, short)       |
|    | OPDEFN(% , L, unsigned short)            | OPDEFN(% , L, signed int)    | OPDEFN(% , L,              |
|    | unsigned int) OPDEFN(% , L, long)        | OPDEFN(% , L, unsigned long) |                            |
|    | OPDEFN(% , L, float)                     | OPDEFN(% , L, double)        | OPDEFN(% , L,              |
|    | long double)                             |                              |                            |
| 45 | OPDEFN(% , bool,L)                       | OPDEFN(% , char,L)           |                            |
|    | OPDEFN(% , const char *,L)               | OPDEFN(% , signed char,      | L) OPDEFN(% ,              |
|    | unsigned char,L)                         | OPDEFN(% , short,L)          |                            |
|    | OPDEFN(% , unsigned short,L)             | OPDEFN(% , signed int,L)     |                            |
|    | OPDEFN(% , unsigned int,                 | L) OPDEFN(% , long,L)        |                            |
| 50 | OPDEFN(% , unsigned long,L)              | OPDEFN(% , float,L)          |                            |
|    | OPDEFN(% , double,                       | L) OPDEFN(% , long double,   | L)                         |
|    |                                          |                              |                            |
|    | LargeInt &operator +=(const LargeInt &); |                              |                            |
|    | LargeInt &operator -=(const LargeInt &); |                              |                            |
| 55 | LargeInt &operator *=(const LargeInt &); |                              |                            |
|    | LargeInt &operator /=(const LargeInt &); |                              |                            |
|    | LargeInt &operator %=(const LargeInt &); |                              |                            |

```

//Use this to get the result and remainder at the same time,
//since they are found concurrently in the divide operation. This
//uses a fast, recursive, divide and conquer algorithm I devised.
5 friend LargeInt Divide(const LargeInt &, const LargeInt &,
 LargeInt &);

//*** Unary ***
//These operations are all optimized for speed and efficiency.
10 friend LargeInt operator +(const LargeInt &);
friend LargeInt operator -(const LargeInt &);
LargeInt &operator ++(); //prefix. Prefix is faster than
//postfix. Use this instead when possible.
15 LargeInt operator ++(int); //postfix.
LargeInt &operator --();
LargeInt operator --(int);
//One's compliment.
friend LargeInt operator ~(const LargeInt &);
20 //Two's compliment. ++(~X) will only work as long as you remain
//in the same Fixed Bit Length, since ++ isn't a bitwise
//operator. The (-) operator is also the equivalent of Two's
//compliment, except that it keeps the number in signed format.
//This function returns an unsigned fixed bit result instead.
25 friend LargeInt Twos(const LargeInt &);

//*** Binary ***

30 OPDEFN(&, L, L) OPDEFN(&, L, bool)
OPDEFN(&, L, char) OPDEFN(&, L, const char *)
OPDEFN(&, L, signed char) OPDEFN(&, L, unsigned char) OPDEFN(&, L,
short) OPDEFN(&, L, unsigned short)
OPDEFN(&, L, signed int) OPDEFN(&, L, unsigned int) OPDEFN(&, L,
long) OPDEFN(&, L, unsigned long)
35 OPDEFN(&, L, float) OPDEFN(&, L, double) OPDEFN(&, L,
long double) OPDEFN(&, bool, L) OPDEFN(&, char, L)
OPDEFN(&, const char *, L)
OPDEFN(&, signed char, L) OPDEFN(&, unsigned char, L) OPDEFN(&, short, L)
OPDEFN(&, unsigned short, L)
40 OPDEFN(&, signed int, L) OPDEFN(&, unsigned int, L) OPDEFN(&,
long, L) OPDEFN(&, unsigned long, L)
OPDEFN(&, float, L) OPDEFN(&, double, L)
OPDEFN(&, long double, L) OPDEFN(|, L, L)
OPDEFN(|, L, bool) OPDEFN(|, L, char) OPDEFN(|, L,
45 const char *) OPDEFN(|, L, signed char) OPDEFN(|, L, unsigned char) OPDEFN(|, L,
short) OPDEFN(|, L, unsigned short) OPDEFN(|, L, signed int)
OPDEFN(|, L, unsigned int) OPDEFN(|, L, long) OPDEFN(|, L,
unsigned long) OPDEFN(|, L, float) OPDEFN(|, L, double)
OPDEFN(|, L, long double)
50 OPDEFN(|, bool, L) OPDEFN(|, char, L)
OPDEFN(|, const char *, L) OPDEFN(|, signed char, L) OPDEFN(|, unsigned
char, L) OPDEFN(|, short, L)
OPDEFN(|, unsigned short, L) OPDEFN(|, signed int, L)
OPDEFN(|, unsigned int, L) OPDEFN(|, long, L)
55 OPDEFN(|, unsigned long, L) OPDEFN(|, float, L)
OPDEFN(|, double, L) OPDEFN(|, long double, L)

```

```

 OPDEFN(^, L, L) OPDEFN(^, L, bool)
 OPDEFN(^, L, char) OPDEFN(^, L, const char *)
 OPDEFN(^, L, signed char) OPDEFN(^, L, unsigned char) OPDEFN(^, L,
5 short) OPDEFN(^, L, unsigned short)
 OPDEFN(^, L, signed int) OPDEFN(^, L, unsigned int) OPDEFN(^, L,
long) OPDEFN(^, L, unsigned long)
 OPDEFN(^, L, float) OPDEFN(^, L, double) OPDEFN(^, L,
long double) OPDEFN(^, bool, L) OPDEFN(^, char, L)
 OPDEFN(^, const char *, L)
10 OPDEFN(^, signed char, L) OPDEFN(^, unsigned char, L) OPDEFN(^, short,
L) OPDEFN(^, unsigned short, L)
 OPDEFN(^, signed int, L) OPDEFN(^, unsigned int, L)
 OPDEFN(^, long, L) OPDEFN(^, unsigned long, L)
 OPDEFN(^, float, L) OPDEFN(^, double, L)
15 OPDEFN(^, long double, L) OPDEFN(<<, L, L) OPDEFN(<<, L,
bool) OPDEFN(<<, L, char) OPDEFN(<<, L, const char *) OPDEFN(<<, L,
signed char) OPDEFN(<<, L, unsigned char) OPDEFN(<<, L, short) OPDEFN(<<, L,
unsigned short) OPDEFN(<<, L, signed int) OPDEFN(<<, L, unsigned int) OPDEFN(<<, L,
long) OPDEFN(<<, L, unsigned long) OPDEFN(<<, L, float)
20 OPDEFN(<<, L, double) OPDEFN(<<, L, long double)
 OPDEFN(<<, bool, L) OPDEFN(<<, char, L)
 OPDEFN(<<, const char *, L) OPDEFN(<<, signed char, L) OPDEFN(<<, unsigned
char, L) OPDEFN(<<, short, L)
 OPDEFN(<<, unsigned short, L) OPDEFN(<<, signed int, L) OPDEFN(<<, unsigned
25 int, L) OPDEFN(<<, long, L)
 OPDEFN(<<, unsigned long, L) OPDEFN(<<, float, L)
 OPDEFN(<<, double, L) OPDEFN(<<, long double, L)
 OPDEFN(>>, L, L) OPDEFN(>>, L, bool)
 OPDEFN(>>, L, char) OPDEFN(>>, L, const char *)
30 OPDEFN(>>, L, signed char) OPDEFN(>>, L, unsigned char) OPDEFN(>>, L, short)
 OPDEFN(>>, L, unsigned short)
 OPDEFN(>>, L, signed int) OPDEFN(>>, L, unsigned int) OPDEFN(>>, L,
long) OPDEFN(>>, L, unsigned long)
 OPDEFN(>>, L, float) OPDEFN(>>, L, double)
35 OPDEFN(>>, L, long double) OPDEFN(>>, bool, L)
 OPDEFN(>>, char, L) OPDEFN(>>, const char *, L)
 OPDEFN(>>, signed char, L) OPDEFN(>>, unsigned char, L)
 OPDEFN(>>, short, L) OPDEFN(>>, unsigned short, L)
 OPDEFN(>>, signed int, L) OPDEFN(>>, unsigned int, L)
40 OPDEFN(>>, long, L) OPDEFN(>>, unsigned long, L)
 OPDEFN(>>, float, L) OPDEFN(>>, double, L)
 OPDEFN(>>, long double, L)

 LargeInt &operator &=(const LargeInt &);
45 LargeInt &operator |=(const LargeInt &);
 LargeInt &operator ^=(const LargeInt &);
 LargeInt &operator <=(const LargeInt &);
 LargeInt &operator >=(const LargeInt &);

50 //*** Comparison ***
 #undef OPDEFN
 #define OPDEFN(Op, LType, RType) friend bool operator Op(LType, RType);

 OPDEFN(<, L, L) OPDEFN(<, L, bool)
55 OPDEFN(<, L, char) OPDEFN(<, L, const char *)
 OPDEFN(<, L, signed char) OPDEFN(<, L, unsigned char) OPDEFN(<, L,
short) OPDEFN(<, L, unsigned short)

```

|                   |                                       |                               |                      |
|-------------------|---------------------------------------|-------------------------------|----------------------|
|                   | OPDEFN(<, L, signed int)              | OPDEFN(<, L, unsigned int)    | OPDEFN(<, L,         |
| long)             | OPDEFN(<, L, unsigned long)           |                               |                      |
|                   | OPDEFN(<, L, float)                   | OPDEFN(<, L, double)          | OPDEFN(<, L,         |
| long double)      | OPDEFN(<, bool, L)                    |                               | OPDEFN(<, char,      |
| 5                 | L) OPDEFN(<, const char *, L)         |                               |                      |
|                   | OPDEFN(<, signed char, L)             | OPDEFN(<, unsigned char, L)   |                      |
|                   | OPDEFN(<, short, L)                   | OPDEFN(<, unsigned short, L)  |                      |
|                   | OPDEFN(<, signed int, L)              | OPDEFN(<, unsigned int, L)    |                      |
|                   | OPDEFN(<, long, L)                    | OPDEFN(<, unsigned long, L)   |                      |
| 10                | OPDEFN(<, float, L)                   | OPDEFN(<, double, L)          |                      |
|                   | OPDEFN(<, long double, L)             | OPDEFN(>, L, L)               |                      |
|                   | OPDEFN(>, L, bool)                    | OPDEFN(>, L, char)            | OPDEFN(>, L,         |
| const char *)     | OPDEFN(>, L, signed char)             | OPDEFN(>, L, unsigned char)   | OPDEFN(>, L,         |
| short)            | OPDEFN(>, L, unsigned short)          | OPDEFN(>, L, signed int)      |                      |
| 15                | OPDEFN(>, L, unsigned int)            | OPDEFN(>, L, long)            | OPDEFN(>, L,         |
| unsigned long)    | OPDEFN(>, L, float)                   | OPDEFN(>, L, double)          |                      |
|                   | OPDEFN(>, L, long double)             |                               |                      |
|                   | OPDEFN(>, bool, L)                    | OPDEFN(>, char, L)            |                      |
| 20                | OPDEFN(>, const char *, L)            | OPDEFN(>, signed char, L)     |                      |
|                   | OPDEFN(>, unsigned char, L)           | OPDEFN(>, short, L)           |                      |
|                   | OPDEFN(>, unsigned short, L)          | OPDEFN(>, signed int, L)      |                      |
|                   | OPDEFN(>, unsigned int, L)            | OPDEFN(>, long, L)            |                      |
|                   | OPDEFN(>, unsigned long, L)           | OPDEFN(>, float, L)           |                      |
| 25                | OPDEFN(>, double, L)                  | OPDEFN(>, long double, L)     |                      |
|                   | OPDEFN(<=, L, L)                      | OPDEFN(<=, L, bool)           |                      |
|                   | OPDEFN(<=, L, char)                   | OPDEFN(<=, L, const char *)   |                      |
|                   | OPDEFN(<=, L, signed char)            | OPDEFN(<=, L, unsigned char)  | OPDEFN(<=, L, short) |
|                   | OPDEFN(<=, L, unsigned short)         |                               |                      |
|                   | OPDEFN(<=, L, signed int)             | OPDEFN(<=, L, unsigned int)   | OPDEFN(<=, L,        |
| 30                | long) OPDEFN(<=, L, unsigned long)    |                               |                      |
|                   | OPDEFN(<=, L, float)                  | OPDEFN(<=, L, double)         |                      |
|                   | OPDEFN(<=, L, long double)            | OPDEFN(<=, bool, L)           |                      |
|                   | OPDEFN(<=, char, L)                   | OPDEFN(<=, const char *, L)   |                      |
|                   | OPDEFN(<=, signed char, L)            | OPDEFN(<=, unsigned char, L)  |                      |
| 35                | OPDEFN(<=, short, L)                  | OPDEFN(<=, unsigned short, L) |                      |
|                   | OPDEFN(<=, signed int, L)             | OPDEFN(<=, unsigned int, L)   |                      |
|                   | OPDEFN(<=, long, L)                   | OPDEFN(<=, unsigned long, L)  |                      |
|                   | OPDEFN(<=, float, L)                  | OPDEFN(<=, double, L)         |                      |
|                   | OPDEFN(<=, long double, L)            | OPDEFN(>=, L, L)              | OPDEFN(>=, L,        |
| 40                | bool) OPDEFN(>=, L, char)             | OPDEFN(>=, L, const char *)   |                      |
|                   | OPDEFN(>=, L, signed char)            | OPDEFN(>=, L, unsigned char)  | OPDEFN(>=, L, short) |
|                   | OPDEFN(>=, L, unsigned short)         | OPDEFN(>=, L, signed int)     | OPDEFN(>=, L,        |
| unsigned int)     | OPDEFN(>=, L, long)                   | OPDEFN(>=, L, unsigned long)  |                      |
|                   | OPDEFN(>=, L, float)                  | OPDEFN(>=, L, double)         | OPDEFN(>=, L,        |
| 45                | long double)                          |                               |                      |
|                   | OPDEFN(>=, bool, L)                   | OPDEFN(>=, char, L)           |                      |
|                   | OPDEFN(>=, const char *, L)           | OPDEFN(>=, signed char, L)    | OPDEFN(>=,           |
| unsigned char, L) | OPDEFN(>=, short, L)                  | OPDEFN(>=, unsigned short, L) |                      |
|                   | OPDEFN(>=, signed int, L)             | OPDEFN(>=, unsigned int, L)   | OPDEFN(>=,           |
| 50                | long, L) OPDEFN(>=, unsigned long, L) | OPDEFN(>=, float, L)          |                      |
|                   | OPDEFN(>=, double, L)                 | OPDEFN(>=, long double, L)    |                      |
|                   | OPDEFN(==, L, L)                      | OPDEFN(==, L, bool)           |                      |
|                   | OPDEFN(==, L, char)                   | OPDEFN(==, L, const char *)   |                      |
|                   | OPDEFN(==, L, signed char)            | OPDEFN(==, L, unsigned char)  | OPDEFN(==, L, short) |
| 55                | OPDEFN(==, L, unsigned short)         |                               |                      |
|                   | OPDEFN(==, L, signed int)             | OPDEFN(==, L, unsigned int)   | OPDEFN(==, L,        |
| long)             | OPDEFN(==, L, unsigned long)          |                               |                      |

```

5 OPDEFN(==, L, float) OPDEFN(==, L, double)
 OPDEFN(==, L, long double) OPDEFN(==, bool, L)
 OPDEFN(==, char, L) OPDEFN(==, const char *, L)
 OPDEFN(==, signed char, L) OPDEFN(==, unsigned char, L)
 OPDEFN(==, short, L) OPDEFN(==, unsigned short, L)
 OPDEFN(==, signed int, L) OPDEFN(==, unsigned int, L)
 OPDEFN(==, long, L) OPDEFN(==, unsigned long, L)
 OPDEFN(==, float, L) OPDEFN(==, double, L) OPDEFN(==,
10 long double, L) OPDEFN(!=, L, L) OPDEFN(!=, L, bool)
 OPDEFN(!=, L, char) OPDEFN(!=, L, const char *) OPDEFN(!=, L,
signed char) OPDEFN(!=, L, unsigned char) OPDEFN(!=, L, short) OPDEFN(!=, L,
unsigned short) OPDEFN(!=, L, signed int) OPDEFN(!=, L, unsigned int) OPDEFN(!=, L,
long) OPDEFN(!=, L, unsigned long) OPDEFN(!=, L, float)
 OPDEFN(!=, L, double) OPDEFN(!=, L, long double)
15 OPDEFN(!=, bool, L) OPDEFN(!=, char, L)
 OPDEFN(!=, const char *,L) OPDEFN(!=, signed char, L) OPDEFN(!=,
unsigned char,L) OPDEFN(!=, short, L) OPDEFN(!=, unsigned short,L)
 OPDEFN(!=, signed int, L) OPDEFN(!=, unsigned int,L) OPDEFN(!=, long,
L) OPDEFN(!=, unsigned long,L) OPDEFN(!=, float, L)
20 OPDEFN(!=, double,L) OPDEFN(!=, long double, L)
 OPDEFN(&&, L, L) OPDEFN(&&, L, bool)
 OPDEFN(&&, L, char) OPDEFN(&&, L, const char *)
 OPDEFN(&&, L, signed char) OPDEFN(&&, L, unsigned char) OPDEFN(&&, L, short)
 OPDEFN(&&, L, unsigned short)
25 OPDEFN(&&, L, signed int) OPDEFN(&&, L, unsigned int) OPDEFN(&&,
L, long) OPDEFN(&&, L, unsigned long)
 OPDEFN(&&, L, float) OPDEFN(&&, L, double)
 OPDEFN(&&, L, long double) OPDEFN(&&, bool, L)
30 OPDEFN(&&, char, L) OPDEFN(&&, const char *, L)
 OPDEFN(&&, signed char, L) OPDEFN(&&, unsigned char, L)
 OPDEFN(&&, short, L) OPDEFN(&&, unsigned short, L)
 OPDEFN(&&, signed int, L) OPDEFN(&&, unsigned int, L)
 OPDEFN(&&, long, L) OPDEFN(&&, unsigned long, L)
 OPDEFN(&&, float, L) OPDEFN(&&, double, L)
35 OPDEFN(&&, long double, L) OPDEFN(||, L, L) OPDEFN(||, L,
bool) OPDEFN(||, L, char) OPDEFN(||, L, const char *)
 OPDEFN(||, L, signed char) OPDEFN(||, L, unsigned char) OPDEFN(||, L, short)
 OPDEFN(||, L, unsigned short) OPDEFN(||, L, signed int) OPDEFN(||, L,
unsigned int) OPDEFN(||, L, long) OPDEFN(||, L, unsigned long)
40 OPDEFN(||, L, float) OPDEFN(||, L, double) OPDEFN(||, L, long
double)
 OPDEFN(||, bool, L) OPDEFN(||, char, L)
 OPDEFN(||, const char *,L) OPDEFN(||, signed char, L) OPDEFN(||,
unsigned char,L) OPDEFN(||, short, L) OPDEFN(||, unsigned short,L)
45 OPDEFN(||, signed int,L) OPDEFN(||, unsigned int,L) OPDEFN(||, long,
L)
 OPDEFN(||, unsigned long,L) OPDEFN(||, float, L)
 OPDEFN(||, double, L) OPDEFN(||, long double, L)

50 //Use if(X) or if(!X) instead of comparing to zero.
 friend bool operator !(const LargeInt &);

 #undef OPDEFN
 #undef L

55 //*** Stream Operators ***

```

[illegible]